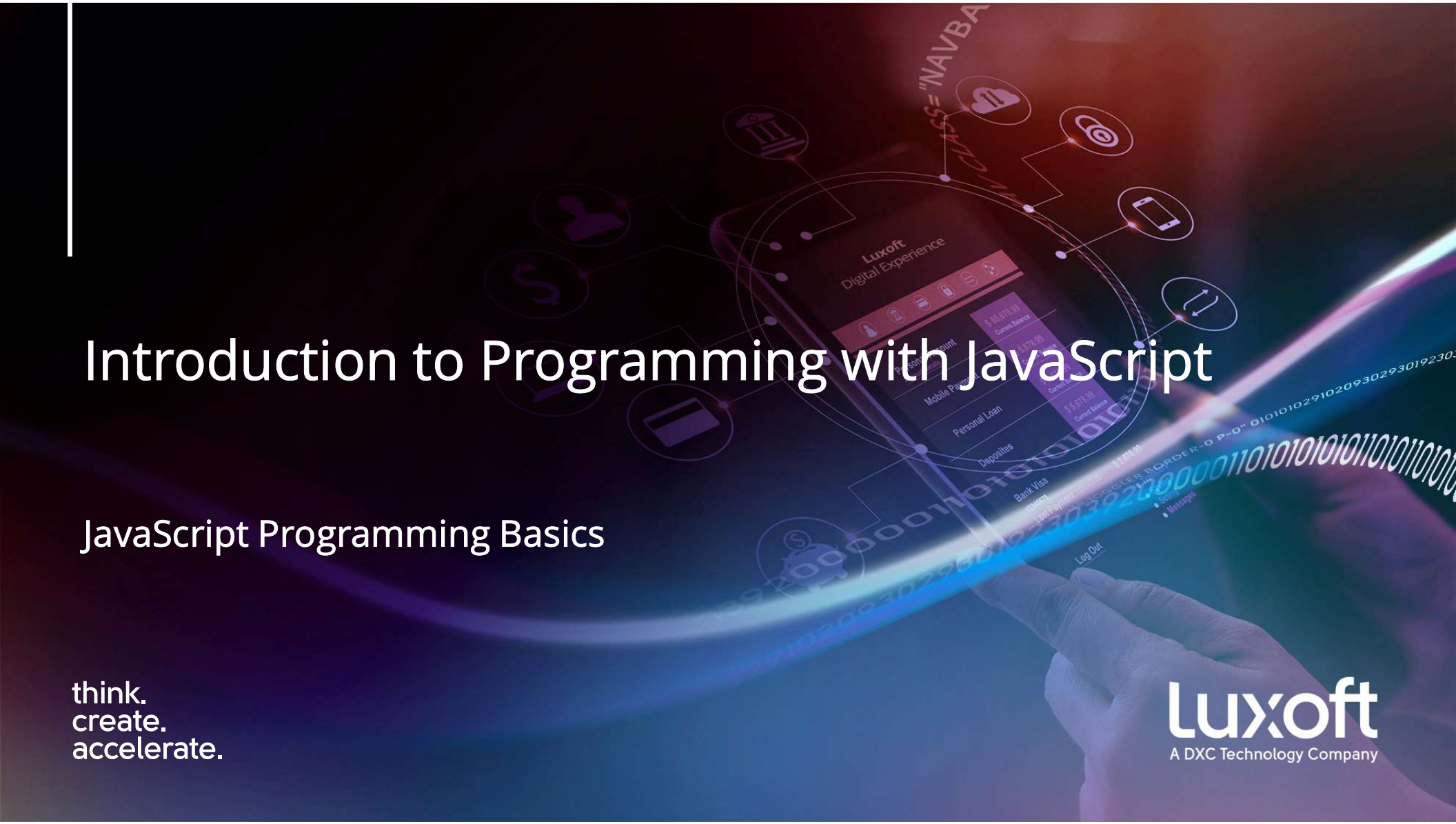


Introduction to Programming with JavaScript

JavaScript Programming Basics

think.
create.
accelerate.

Luxoft
A DXC Technology Company

What Is JavaScript?

- JavaScript is a programming language designed to add interactivity to web pages.
- Lately, it has become one of the leading languages for developing full-scale web applications.
- JavaScript is a language that can be compiled by browsers on-the-go.

Adding a script to the page

- To execute JS code, we can simply put it in a `<script>` tag in HTML.
- Just like with CSS, it is considered cleaner to create separate .js files that contain the code.
- To do that, just add an attribute to the script tag like this:

```
<script src="script.js"></script>
```

Alerts and Logs

- There are two commands of simple outputs of your JS code - **`alert`** and **`console.log`**
- The first one creates a dialog window
- The second one prints the output into the Console tab of the browser

Task

- Create a new file with a .js extension - script.js.
- Inside, write the following code:

```
alert('Hello world!');
```

- Include the .js file in the HTML file using the **<script>** tag.
- Open the page in browser - a popup message will greet you.
- Replace the alert with the following code:

```
console.log('Hello world!');
```

- Refresh the page in browser and open the console (F12).

Statements

- A computer program is a list of "instructions" to be "executed" by a computer.
- In a programming language, these programming instructions are called statements.

```
let x, y, z;    // Statement 1
x = 5;          // Statement 2
y = 6;          // Statement 3
z = x + y;      // Statement 4
```

Variables in JavaScript

- JavaScript features a dynamic typing - all variables can be declared without specifying their type
- Variables are declared using the keywords **`let`** and **`const`**;
- Variables can be assigned a value using = operator;
- Const is preferred, use **`const`**, if you don't need to re-assign the value. With **`const`** you need to assign the value in the same statement.
- Use **`let`**, if you need to reassign the value later.
  - **`const piNumber = 3.14;`**
  - **`let someValue; // declare the variable`**
  - **`someValue = 1; // assign the value later`**

Common Operators

- In the console of the browser, experiment with the usage of common operators like:
  - `1 + 2`
  - `1 % 2`
  - `"abc" + "def"`
  - `true && false`

Arithmetic Operators

Operator	Operator
+	+=
-	-=
*	*=
/	/=
%	%=
++	--

Arithmetic Operators

```
a = 1+1;
b = a*3;
c = b / 4;
d = b - a;
e = -d;
d +=a;
c += b++;

c += ++b;

f = c % b;
```

```
a = 2
b = 6
c = 1
d = 4
e = -4
d = 6
c = 7
b = 7
b = 8
c = 15
f = 7
```

Operators precedence (priority)

14	unary plus	+
14	unary negation	-
13	exponentiation	**
12	multiplication	*
12	division	/
11	addition	+
11	subtraction	-
…	…	…
2	assignment	=
…	…	…

Logical Operators

Operator	Expression
AND (&&)	expr1 && expr2
OR (||)	expr1 || expr2
NOT (!)	!expr

Logical Operators

```
a = true;
b = false;
c = a || b;
d = a && b;
e = (!a && b) || (a && !b);
f = !a;
```

```
a = true
b = false
c = true
d = false
e = true
f = false
```

Boolean Values

- **`true`** or **`false`**
- The Boolean() function finds out if an expression (or a variable) is true

```
Boolean(10 > 9);
```

Everything with a Value is true

```
100
3.14
-15
"Hello"
"false"
7 + 1 + 3.14
```

Everything without a Value is false

```
0
-0
""
null
False

let x = 10 / "abc"; //NaN
Boolean(x); //NaN is false
```

Comparison Operators

Operator	Expression
Equality (==)	x == y
Inequality (!=)	x != y
More (>)	x > y
More or equal (>=)	x >= y
Less (<)	x < y
Less or equal (<=)	x <= y
Equal, no types conversion (===)	x === y
Not equal, no types conversion (!==)	x !== y

Comparison Operators

```
a = 4;
b = 1;
c = a < b;
d = a == b;

b = “4”;
d = a == b;
d = a === b;
```

```
c = false
d = false



d = true
d = false
```

Task

- Using the console:
  - Experiment with the examples of operators, as presented on the previous slides

Data Comparison in JavaScript

- Due to the dynamic types, it is often hard to compare the values of variables.
- There are two operators for comparison:

  == compares values

  === compares values and types

Task

- Using the console:
  - Create a new variable named **`myName`** and assign your name as a string value to it.
  - Add a console command to write **`"Hello" + myName`** so that the page would greet you by name.

Accessing HTML

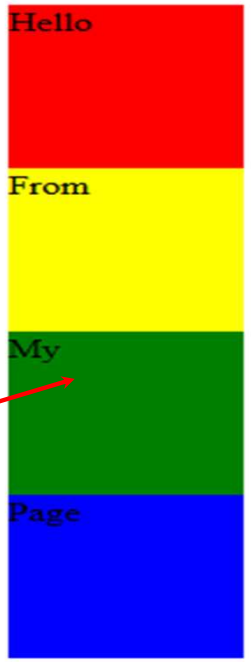
- JavaScript intends to manipulate the document - so a bunch of keywords are reserved for that
- The keyword **document** allows you to get HTML elements and manipulate them

Accessing HTML

```
let a = document.getElementById("someId");
let b = document.getElementByClassName("someClass");
let e = document.getElementsByTagName("div");
```

Accessing HTML

```
<div id="first" style="width:100px; height:100px;
  background-color:red;">Hello</div>
<div id="second" style="width:100px; height:100px;
  background-color:yellow;">From</div>
<div id="third" style="width:100px; height:100px;
  background-color:green;">My</div>
<div id="fourth" style="width:100px; height:100px;
  background-color:blue;">Page</div>

let g = document.getElementById("third");
g.innerHTML = "I am green!";
```



accessinghtml.html

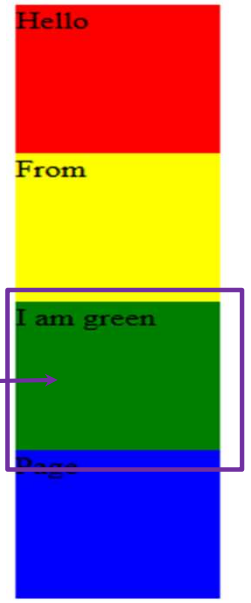
Accessing HTML

```
<div id="first" style="width:100px; height:100px;
  background-color:red;">Hello</div>
<div id="second" style="width:100px; height:100px;
  background-color:yellow;">From</div>
<div id="third" style="width:100px; height:100px;
  background-color:green;">My</div>
<div id="fourth" style="width:100px; height:100px;
  background-color:blue;">Page</div>

let e = document.getElementsByTagName("div");
```

e[0]
e[1]
e[2]
e[3]

Accessing HTML

```
<div id="first" style="width:100px; height:100px;
  background-color:red;">Hello</div>
<div id="second" style="width:100px; height:100px;
  background-color:yellow;">From</div>
<div id="third" style="width:100px; height:100px;
  background-color:green;">My</div>
<div id="fourth" style="width:100px; height:100px;
  background-color:blue;">Page</div>

let e = document.getElementsByTagName("div");
e[2].innerHTML = "I am green";
```



accessinghtml2.html

Task

- Create an HTML file to include a picture from the previous slides.
- Create a script.js file, referenced inside the HTML, that will modify the content of the HTML file as demonstrated.
- Try to change the content of different elements from the HTML in various ways.

Task

- Open the console from the browser and introduce a few simple calculations, as below. Vary them and analyze the results.

```
a=10
b=20
a+b
(a+b) * (b-a)
!a
a++
++a
```

Truthy and falsy

- A truthy value is a value that is considered true when encountered in a Boolean context.
- All values are truthy unless they are defined as falsy (i.e., except for **`false`**, **`0`**, **`-0`**, **`0n`**, **`""`**, **`null`**, **`undefined`**, and **`NaN`**).
- A falsy value is a value that is considered false when encountered in a Boolean context.

Examples of truthy values in JavaScript

```
if (true)
if ({})
if ([])
if (42)
if ("0")
if ("false")
if (new Date())
if (-42)
if (12n)
if (3.14)
if (-3.14)
if (Infinity)
if (-Infinity)
```

Examples of falsy values in JavaScript

```
if (false)
if (null)
if (undefined)
if (0)
if (-0)
if (0n)
if (NaN)
if ("")
```

‘if’ statement (conditional)

```
if (expression) {
    statement1
}
else {
    statement2
}
```

- The **`if`** statement executes **`statement1`** if expression is truthy and executes **`statement2`** if expression is falsy

03example01.html

‘If’ statement (conditional)

```
if (n == 1) {
    console.log("You have 1 new message.");
}
else {
    console.log("You have " + n + " new messages.");
}
```

03example01.html

‘Switch’ statement (conditional)

```
switch (expression) {
  case value1:
    //Statements executed when the
    //result of expression matches value1
    [break;]
  ...
  case valueN:
    //Statements executed when the
    //result of expression matches valueN
    [break;]
  [default:
    //Statements executed when none of
    //the values match the value of the expression
    [break;]]
}
```

[ ] in this pseudo-code means ‘optional’ (‘break’ statements and ‘default’ are not required)

‘Switch’ statement (conditional)

- **`expression`** is evaluated
- Case with a label matching the value of **`expression`** is chosen and statements are executed
- If no matching label, the program looks for the optional default clause
- **`break`** causes the interpreter to jump to the end of the **`switch`** and continue with the statement that follows it
- The matching case is determined using the === identity operator, not the == equality operator

Task

- Change the code from 03example01.html and 03script01.js to replace the **`if`** statement with a **`switch`** statement with one **`case`** label and the **`default`** label.
- Extend the **`switch`** statement with more **`case`** labels.

‘While’ statement (‘while’ loop)

```
while (expression) {
    statement
}
```

- **`expression`** is evaluated
- If **`expression`** is falsy, the interpreter skips the statement
- If **`expression`** is truthy, the interpreter executes the statement and jumps to the top of the loop, evaluating **`expression`** again

```
let count = 0;
while (count < 10) {
    console.log(count);
    count++;
}
```

03example02.html

‘do/while’ statement (‘do/while’ loop)

```
do {
    statement
} while (expression);
```

- The do/while loop is like a while loop, except that the loop expression is tested at the bottom. The body of the loop is always executed at least once.
- The do/while loop must always be terminated with a semicolon.

‘for’ statement (‘for’ loop)

```
for(initialize ; test ; increment) {
    statement
}
```

- **`initialize`**, **`test`**, and **`increment`** are 3 expressions for initializing, testing, and incrementing the loop variable
- **`initialize`** is evaluated once, before the loop begins
- **`test`** is evaluated before each iteration, which controls whether the body of the loop is executed
- **`increment`** is evaluated at the end of the loop

‘for’ statement (‘for’ loop)

```
let i, j, sum = 0;
for(i = 0, j = 10 ; i < 10 ; i++, j--) {
    sum += i * j;
}
```

03example03.html

Task

- Change the code from 03example02.html and 03script02.js to replace the **`while`** loop with a **`do.. while`** loop and a **`for`** loop.
- Change the code from 03example03.html and 03script03.js to replace the **`for`** loop with a **`do.. while`** loop and a **`while`** loop.

Function Definition

- Function definitions begin with the keyword **`function`** followed by:
  - An identifier that names the function
  - A pair of parentheses around a comma-separated list of zero or more identifiers
  - A pair of curly braces with zero or more JavaScript statements inside

```
function <name>(<param>, <param>, ...) {
    <statements>
}
```

Function Definition

- The function may contain a **`return`** statement
- The **`return`** statement stops the function and returns the value of its expression (if any) to the caller
- If **`return`** does not have an associated expression, it returns **`undefined`**
- If a function does not contain **`return`**, it executes each statement in the body and returns **`undefined`**

Function Definition

```
// The distance between Cartesian points (x1,y1) and (x2,y2).
function distance(x1, y1, x2, y2) {
    let dx = x2 - x1;
    let dy = y2 - y1;
    return Math.sqrt(dx*dx + dy*dy);
}

// Recursive function
function factorial(x) {
    if (x <= 1) return 1;
    return x * factorial(x-1);
}
```

03example04.html

Function Definition

- In JavaScript, functions may be nested within other functions.

```
function hypotenuse(a, b) {
    function square(x) { return x*x; }
    return Math.sqrt(square(a) + square(b));
}
```

- Function definition can appear in global code, or within other functions, but they cannot appear inside of loops or conditionals.

03example04.html

Task

- Create a function to calculate the area of a triangle with sides a, b and c. Use Heron's formula for this.

$$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)}$$

Area = area

s = semi-perimeter

a = length of side a

b = length of side b

c = length of side c

- Test the function by calling it and calculating the area of a few triangles.

Function arguments and arguments object

- A special **`Argument`** object is available inside any function
- The identifier **`arguments`** refers to it
- **`arguments`** has a **`length`** property - the number of elements it contains

```
function f(x, y, z) {
    // Verify that the right number of arguments was passed
    if (arguments.length != 3) {
        //A number of arguments other than 3
    }
    // Now do the actual function...
}
```

03example05.html

Arrow Functions

- Arrow functions allow us to write shorter function syntax.

```
function hello() {
  return "Hello World!";
}

let hello = () => "Hello World!";
```

Arrow Functions

- Arrow functions return a value by default.

```
hello = () => "Hello World!";
```

- Arrow function with parameters:

```
hello = (val) => "Hello " + val;
```

- For one parameter, you can skip the parentheses:

```
hello = val => "Hello " + val;
```

Arrow Functions

```
let myArrowWith1Param = oneParam => {
    console.log(
       'Executing myArrowWith1Param arrow function!');
    return oneParam * 2;
};

console.log(myArrowWith1Param(4));

let myArrowWithParams = (param1, param2) => param1 + param2;
console.log(myArrowWithParams('x', 'y'));
```

03example06.html

Task

- Change the code from **03example04.html** and **03script04.js** to replace the already defined functions with arrow functions.
- Test the functions by calling them with the same arguments as the regular functions were previously called.

Thank you!